

Emulation of Biological Networks in Reconfigurable Hardware

Natasa Miskov-Zivanov¹, Andrew Bresticker², Deepa Krishnaswamy², SreesanVenkatakrishnan²,
Diana Marculescu² and James R. Faeder¹

¹Department of Computational and Systems Biology, School of Medicine, University of Pittsburgh

²Department of Electrical and Computer Engineering, Carnegie Mellon University

E-mail: {nam66,faeder}@pitt.edu, {abrestic,deepakri,sreesanv,dianam}@ece.cmu.edu

ABSTRACT

Models of biological networks have been studied through simulations using a number of software tools. However, the intrinsic disparity between the sequential nature of microprocessor architecture used in software-based simulations and the highly parallel nature of biological systems may result in prohibitively long simulation times. In this work, we adopt an alternative approach to simulation of biological systems using hardware-based emulation. Our results on Boolean network models show that such an approach can provide speedup of 17,000X when compared to existing software simulation approaches.

1. INTRODUCTION

Over the past decade, a number of computational approaches have been proposed for the purpose of modeling and studying biological systems. The methods range from master equations and the Monte-Carlo method [2], ordinary differential equations (ODEs) [3], reaction rule-based models [4], all the way to Boolean networks [1, 5-9]. A modeling framework that uses Boolean variables to represent states of elements of the network and logical rules to describe the element relationships may be appropriate for modeling many biological networks [8, 9]. This method generates predictions of the trajectory of states followed by a control circuit in a cell, which is often sufficient to advance our knowledge significantly. Examples of applications of the Boolean approach are cell lineage commitment [10], cell cycle progression [11], and cell differentiation [12].

With the increase in the size of the network, the complexity of models of biological networks increases rapidly. Moreover, simulations of these networks are executed sequentially on general purpose CPUs, which is in contrast to the highly parallel nature of information flow within biochemical networks. In general, models of biological networks are simulated by assuming an initial state (or distribution of initial states) as the model input, with dynamical trajectories and steady states as the model output. However, the intrinsic disparity between the sequential nature of microprocessor architecture used in software-based simulations and the highly parallel nature of biological systems may result in prohibitively long simulation times [13].

The main goal of this work is to develop a hardware design methodology for *emulating* biological network behavior into Field Programmable Gate Array (FPGA) platforms, which are ideally suited to implement highly parallel architectures. FPGAs are integrated circuits designed to be configured by the customer or designer after manufacturing and hence, “field-programmable.” FPGAs contain programmable logic components called “logic blocks,” and a hierarchy of reconfigurable interconnects that allow the blocks to be “wired together”—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex

combinational functions, or merely simple logic gates like AND and XOR. These logic blocks can be programmed using a hardware description language such as Verilog or VHDL (Very high speed integrated circuits Hardware Description Language). The main advantage of FPGAs is that they are inherently parallel as the logic blocks can be programmed completely by the designer and therefore, FPGA platforms have been traditionally used for hardware emulation of computing systems.

Besides emulation of computing systems, the parallel architecture of FPGAs, can also simulate the basic reaction steps of biological networks and attain simulation rates many orders of magnitude greater than currently available microprocessors. Recently, several hardware-oriented approaches [13-15] to biological network modeling have been developed. They have all focused on the implementation of variants of Gillespie’s stochastic simulation algorithm (SSA) [2]. The authors in [13, 14] implemented the SSA as a single FPGA thread, resulting in speedups of about one order of magnitude compared to a software implementation. More recent work [15] implements multiple simulation threads to achieve greater efficiency. While these implementations do make SSA-based simulations more efficient, applications that use ODE, rule-based and logical modeling approaches do not benefit.

In this work, we provide a framework for efficient and accurate analysis of complex signaling and regulatory networks, since their understanding is of great interest in both medicine and biology. More precisely, the hardware-based emulation framework we are proposing is poised to enable several orders of magnitude speedup (as shown in a case study in Section 4) when compared to classic simulation in silicon. We show how FPGAs can be used to build a digital system circuit that emulates a dynamical, logical model of biological signaling network. Since FPGA platforms are built as arrays of simple configurable logic blocks embedded in a programmable interconnection matrix, they are therefore, ideally suited to implement highly concurrent interaction maps of biological networks and their dynamical models due to the one-to-one mapping between the Boolean logic rules associated with a biological network and logic gates available on an FPGA platform.

In the following sections we first describe preliminaries of the logical modeling approach and then we describe steps of our methodology (shown in Figure 1(a)) for designing hardware for emulating biological networks. We outline here the contributions of our work when compared to both software- and hardware-based approaches to studying signaling networks:

- In comparison to software-based approaches to simulate biological networks, our proposed approach is *efficient*, because **hardware-based emulation** can give better performance than software-based simulation.
- In comparison to hardware-based approaches to studying biological networks, our approach:
 - (i) **Implements the logical (dynamic) model of the network**, while previous approaches only implemented the simulation algorithm for ODE models;
 - (ii) Improves efficiency through **implementation of several copies of the model**, allowing for simultaneous runs and faster comparison of several different implementations of the system, or outcomes of different initial conditions and different scenarios.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM-BCB’11, August 1-3, 2011, Chicago, IL, USA.

Copyright 2011 ACM 1-58113-000-0/00/0010...\$10.00.

This work was supported in part by NIH grant UL1-RR024153 and an NSF Expeditions in Computing grant (award ID 0926181).

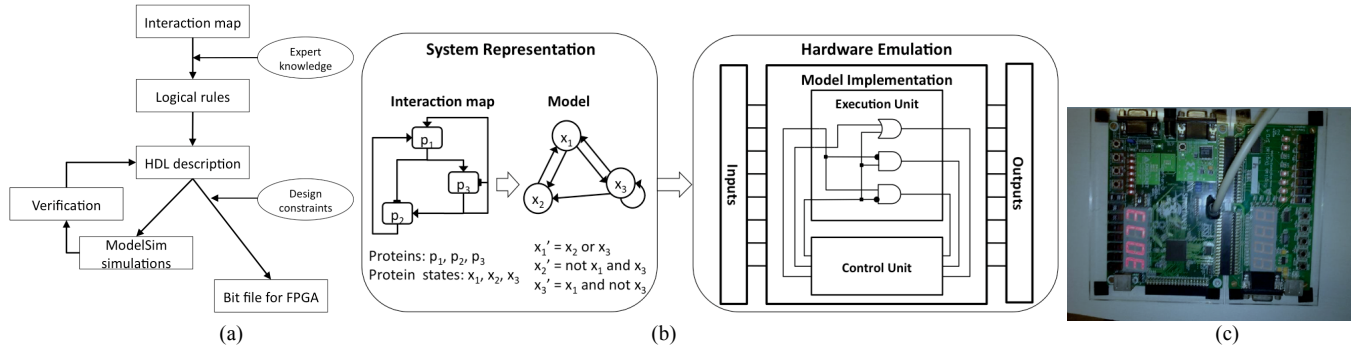


Figure 1. (a) Proposed design methodology steps; (b) Example interaction of three proteins, p_1 , p_2 and p_3 : **Interaction map** defining the direction and sign of interactions, **Model** representation in terms of Boolean network and logic update rules, and main elements of **Hardware Emulation** (**Inputs**, **Outputs**, **Model Implementation**); (c) Picture of an FPGA with LED displays displaying a hexadecimal output.

2. LOGICAL MODELING PRELIMINARIES

In Figure 1(b)(left), an **Interaction map** of a small system with three proteins, p_1 , p_2 and p_3 is shown. We also define the set of variables representing protein states, x_1 , x_2 and x_3 . In this work, we focus on analyzing models that include Boolean or integer variables. In the designed hardware, all variables are eventually represented using Boolean elements. In the past years, several groups have worked on developing Boolean network models for different signaling pathways [1, 6, 7] or gene regulatory networks [11, 16]. These models do not include any information about reaction rates or molecule counts, but instead, they are used to study the dynamics of the system and its behavior (state transitions) from the initial state until reaching a steady state or steady cycle. The model is composed of fixed sets of variables and rules. As shown in the example **Model** in Figure 1(b)(left), each variable has a rule associated with it and the rule defines the next value of a variable (x_1' , x_2' and x_3' in the figure).

Simulations of the model can be either deterministic or stochastic. For example, a deterministic scheme ("synchronous") is to update all variables simultaneously based on the current values of the variables. A stochastic scheme ("asynchronous") is to update variables one at a time in a randomly chosen order. In addition to these two schemes, one can use a combined scheme, where update rules are ranked and, for example, groups of rules are updated in order of their rank while a random asynchronous scheme is applied to all rules within the same rank. Since not all elements of the biological signaling network are updated synchronously, asynchronous and ranked updating schemes are more suitable for this type of networks. In this work, we used only a random asynchronous scheme to obtain results presented in Section 4. The hardware emulation approach that we propose includes all of the schemes described above.

3. HARDWARE IMPLEMENTATION STEPS

In this section, we describe the main steps of our methodology that are presented in Figure 1(a). Once the logical model is defined, the model is used as an input to the hardware design phase. This includes the design of a circuit, which usually starts with defining larger blocks. The next step in hardware design is to define modules within each block and describe them in a Hardware Description Language (HDL). The design can be simulated before the actual implementation in hardware, using an HDL simulator such as ModelSim [17]. The **Hardware Emulation** box in Figure 1(b)(right) presents the main elements of the circuit that is designed to emulate the biological network model:

- **Inputs** - represent external stimuli or environmental effects that can occur anytime during circuit simulation (e.g., changes in conditions outside of the system, inhibitor addition, gene knock-outs, etc.), as well as arrays of initial conditions (e.g., element states or molecule counts).

- **Outputs** - used to collect the information about the behavior of the system as a response to initial conditions and changes in inputs.
- **Model Implementation**, which can consist of two modules:
 - (i) **Execution Unit**, which implements the network (elements and rules);
 - (ii) **Control Unit**, which controls the simulation (and timing of interactions) of the implemented network.
- **Display** - outputs can be directed to a display to visually present the outcomes of model simulations (Figure 1(c)).

When translating biological network models into hardware, one also needs to consider an issue of defining the time of the execution of individual interactions. FPGAs seem very suitable for implementing biological network models because of their parallel nature and in fact, we intend to develop design techniques that can facilitate correct relative timing when emulating signaling networks. One way to approach the problem of implementing asynchronous network updates using synchronous hardware (that uses a clock signal) is to design a system with an embedded pseudo-random number generator, as described in the following.

3.1.1 Circuit design in Verilog

We describe here in more detail our design of the **Model Implementation** module using the hardware description language Verilog, and also show elements of the module in Figure 2(left).

The Execution Unit and Control Unit can be written in Verilog and then synthesized into an FPGA. Our Verilog implementation consists of several different components. The logic rules for all signals in the network are implemented in a combinational logic module called `network_logic`. Here, one can find a one-to-one mapping between the Boolean rule updates from the logical model and the logic assignments in the Verilog code. Three registers are used to represent the current state of the network, the nodes that have been updated in the current round, and the nodes that are fixed at "off" by inhibitors. They are `network_state`, `updated`, and `inhibitors`, respectively. Pseudo-random number generation is performed by a 64-bit Linear Feedback Shift Register (LFSR) module, `lfsr`. The LFSR was chosen for its simple, synthesizable implementation and its long period. The control finite state machine is implemented in the module called `controlpath`. In each round of updating, the least-significant six bits, labeled as `random_addr`, of the output of `lfsr` are used as the random number. This is used as the `select` lines for `network_state` and `updated`.

If `updated[random_addr]` is "off" (0), then the selected node has not been updated this round. Thus, we load the new value of the node represented by `random_addr` into `network_state[random_addr]` and load "on" (1) into `updated[random_addr]`. The inhibitor mask is then applied to the network state. If the selected node has already been updated (`updated[random_addr] = 1`), then we wait until we have an

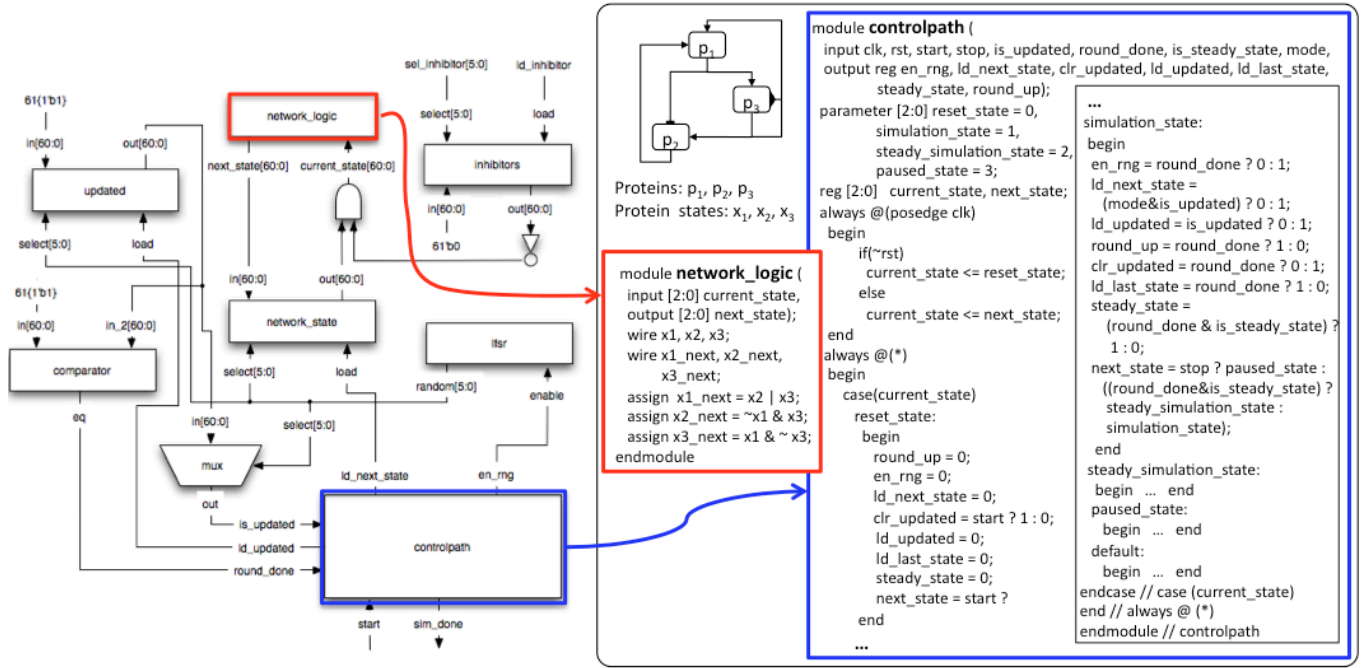


Figure 2. Designed circuit modules (left) and Verilog implementation (right) of two modules: `network_logic` module for the example network from Figure 1(b) and `controlpath` module.

address which corresponds to a node that has not been updated in this round. This continues until all nodes have been updated. That is, `updated[n] = 1` for all signals. At the end of the round, `updated` is cleared and the process continues. A counter is used to count the number of rounds that have been completed. Additionally, one register is used to store the state of the network from the previous round in order to determine if the network has reached a steady state. A simplified model of the circuit (excluding the previous state register and round counter) is shown in Figure 2(left), as well as Verilog code for the `network_logic` module of the example network from Figure 1(b) and the `controlpath` module Figure 2(right).

In order to test different initial conditions and inhibitors, multiple instances of the network can be implemented on an FPGA. We also designed a separate top-level control module to multiplex the inputs and outputs of the separate instances. Each instance could be specified with a different initial state and seed for random number generator.

3.1.2 Circuit simulations using ModelSim

After all blocks of the circuit are designed in hardware description language (Verilog in our case), one can use simulation to analyze the design and to view signal waveforms. ModelSim [17] is a hardware design language simulator that can be used for VHDL, Verilog, or mixed-language simulation. It has an easy-to-use graphical user interface and allows for simulations of behavioral, Register Transfer Level (RTL), and gate-level code separately or simultaneously. ModelSim also supports all FPGA libraries, ensuring accurate timing simulations. It also provides a waveform view of simulated signals, which can be very useful for analyzing element trajectories from initial to steady state.

3.1.3 Design verification

Once the description of a system is finalized, the design needs to be verified. Verification of the implemented model can be approached via functional verification for smaller blocks (*i.e.*, testbench-based simulation), as we have done for the networks that are presented in Section 4. For larger network verification, we plan to employ in the future some of the existing scalable formal

verification tools (either Binary Decision Diagram (BDD)-based [18], Satisfiability (SAT)-based [19], or Model Checking (MC) [20].

3.1.4 Hardware synthesis

Synthesizing the hardware after it is verified should be straightforward. However, this problem in its nature is different from a typical hardware design problem. When designing hardware for the purpose of implementing computer applications, the interplay between power, performance and cost plays a very important role. In the case of designing hardware to emulate biological signaling networks, the minimization of the logic implementation is important from an efficiency perspective, but in our case, one needs to be careful not to make changes that could generate incorrect results. In particular, timing rules imposed on the signaling network should not be undone by the design compilation step. However, if timing rules are properly implemented and constrained by-design in the generated hardware description as described before, typical synthesis tools used in FPGA emulation are not expected to undermine the timing behavior of the signaling networks under consideration.

3.1.5 Displaying results

As an initial way of displaying results, the state of the network is displayed (in hexadecimal digits) on a bank of seven-segment light-emitting diode (LED) displays. Switches are used to select which simulator to display (in case when there are multiple copies of a network on a single FPGA), and which part of that network to display. Switches and buttons were also used to select and load inhibitors, as well as to start and reset the simulations. Using control switches on the FPGA, we can check the results for different ModelSim simulations. For example, we can select a particular inhibitor and load it to the network and observe the change in the steady network state. Figure 1(c) shows an example network state observed on the FPGA board. In the future, we plan to work on approaches to forward outputs to the screen to provide a better visualization of results.

4. RESULTS

We implemented in hardware several existing logical models (T cell large granular lymphocyte leukemia (T-LGL) [1], helper T cell differentiation [21], peripheral naïve T cell differentiation [22]). All

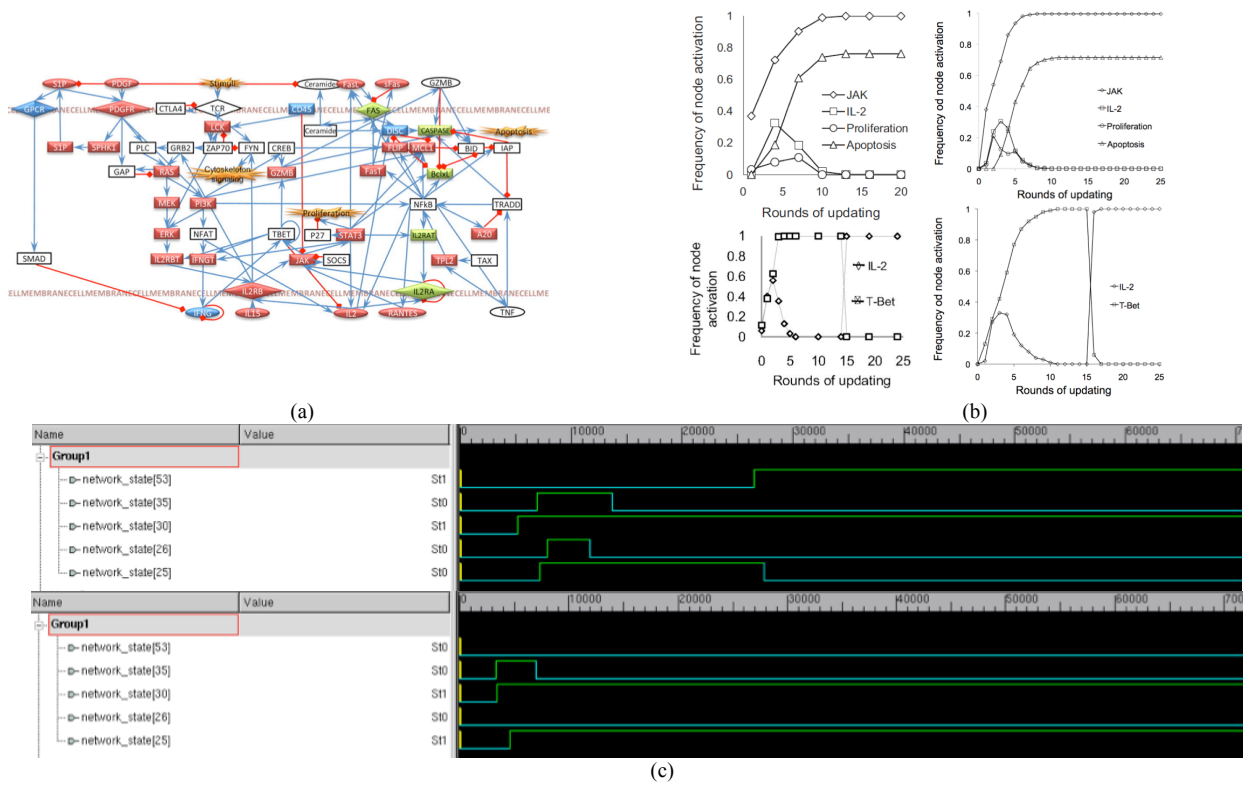


Figure 3. T-LGL model: (a) Interaction map of the system modeled in [1]; (b) Results for *Apoptosis*, *IL-2*, *JAK* and *Proliferation* obtained with software simulations in BooleanNet [1] (top left) and results of the FPGA simulations in ModelSim (top right) and results for *IL-2* and *T-Bet* obtained with software simulations in ModelSim (bottom right); (c) snapshot of *Apoptosis*, *IL-2*, *JAK*, *Proliferation* and *T-Bet* (network_state 53, 35, 30, 26, and 25, respectively) waveforms for two simulations in ModelSim.

models use discrete variables to represent elements of the network and logic update functions to represent interactions between elements. We first implemented these models in a hardware description language, Verilog. The stochasticity that exists in biological networks has been introduced through random ordering of updates of all elements. To compare hardware emulation results with software simulations, we ran simulations of models using BooleanNet [1, 5]. After verifying the design through software simulation, it was synthesized for implementation on a Xilinx FPGA [23].

Due to space constraints, we present here only details of the T-LGL model emulation. We first briefly describe the network model, then we outline main aspects of its hardware implementation and finally, we compare software and hardware results. At the end of this section, we also provide analysis of the speedup obtained through Verilog modeling and hardware emulation, when compared with software simulations in Python using BooleanNet.

4.1 Case study: T-LGL model

In this case study, we present the analysis of signaling network that determines the survival of cytotoxic T lymphocytes (CTL) in T cell large granular lymphocyte (T-LGL) leukemia, which has been described in [1] using a Boolean model.

4.1.1 Model

To systematically understand signaling components that determine the survival of CTL in T-LGL leukemia, the T-LGL survival signaling network was constructed in [1] by integrating the signaling pathways involved in normal CTL activation and the known deregulations of survival signaling in leukemic T-LGL. The interaction map of the T-LGL network from Zhang et al. [1] is shown in Figure 3(a). The overall network contains 61 elements and thus, 61 variables are defined in the model.

As described in Section 2, model simulations are performed using asynchronous update scheme. In these simulations, the node *IL15*

was fixed at "on", while the states of *PDGF*, *Stimuli*, and *Stimuli2* randomly alternated between "on" and "off" throughout the simulation. Furthermore, *IAP*, *MCL1*, *BclxL*, and *IL2RBT* were initially "on". Simulations continue until a fixed number of rounds have been completed or the network has reached a steady state (e.g., the *Apoptosis* signal is "on").

4.1.2 Implementation

Once the network is described using Verilog, we ran simulations with the ModelSim SE Verilog simulator on GNU/Linux. 200 simulations were run, each consisting of 20 rounds of updating following the procedure described in Sections Section 2 and 3.1.1. The LFSR was seeded randomly for each simulation. Through experimentation, it was determined that a maximum of five concurrent simulators could be placed on the given FPGA. The clock for the sequential circuit is tied to the internal 50MHz clock.

4.1.3 Results

A chart showing the activation frequency obtained in [1] for the nodes representing *Apoptosis*, *IL2*, *JAK*, and *Proliferation* is shown in Figure 3(b) (top left). *Apoptosis* occurs at a rate of 0.715, while *JAK*, which is fixed at "on" once activated, has a frequency of activation of 0.995. The frequencies shown in Figure 3(b) (top right) closely match the results obtained by Zhang et al. [1]. We noticed a small difference between the logical model simulations and emulation that results in transient changes of the *Proliferation* variable. This disparity between simulation and emulation may stem from the implementation of a random number generator (LFSR). In Figure 3(b) (bottom charts), we present results for *T-Bet* and *IL-2*, when *T-Bet* is first overexpressed, and then inhibited after 15 rounds. Figure 3(b) (bottom left) shows results from [1] and Figure 3(b) (bottom right) shows the graph that we obtained simulating our design.

In Figure 3(c), we present two sets of waveforms for *Apoptosis*, *IL-2*, *JAK*, *Proliferation* and *T-Bet* (network_state 53, 35, 30, 26,

Table I. Hardware emulation speedup (single instance, 200 runs).

Case study	# nodes	# rounds	BooleanNet	FPGA
I: T-LGL	61	15	60s	0.019s
II: Th diff.	23	25	65s	0.022s
III: T cell diff.	54	16	45s	0.016s

and 25 in the figure, respectively), each obtained with a single simulation in ModelSim. These waveforms represent the trajectories of these variables from the same initial state until the system reaches steady state. As it can be seen from Figure 3(c), different simulation outcomes are possible, due to the fact that stochasticity is included in simulations through the random order of rule updates.

4.2 Hardware emulation speedup

In this section, we discuss the speedup that we observed in our hardware emulation of the three biological networks, when compared to software simulations. In Table I, we outline the speedup data from the three studies.

In Case study I, the runtime for 200 software simulations, with 16 update rounds in each simulation, was approximately 60s. Given the clock frequency for the implemented systems (50MHz), number of rounds necessary to reach the steady state (16), number of steps in each round (61), and number of clock cycles per step (approximately 5) and number of simulations (200), the estimated time for obtaining steady state values for the emulated network is 19.2 ms. In other words, the speedup from a single model implementation, when compared to software simulations in Python was approximately 3100X. Given the fact that multiple copies of the model can be implemented on a single FPGA and run in parallel, the speedup can scale further (possibly, linearly) with the number of model copies on the chip. In the case of the FPGA circuit used in Case study I, five copies were implemented, **providing an overall speedup of 15,000X when compared to software simulations.**

Similar computations to those done in Case study I, were conducted for the Case study II, where the model consists of 23 elements and therefore, each round consists of 23 update steps. In the Case study III, 54 nodes are updated in one round in a random order. We implemented six instances of the network on an FPGA board simultaneously and executed them in parallel. This in turn indicates six times speedup to that estimated by ModelSim for the implementation of a single network instance and approximately **17,000X speedup when compared to software simulations in BooleanNet.**

We are currently working on the implementation of a logical simulator in C under OpenCL. Our preliminary results on an 8-way multi-core platform running Ubuntu Linux on a virtual machine where only four of the eight cores are used show that simulations of the model in Case study I take approximately 0.14s. In other words, our 50MHz FPGA still provides an order of magnitude of speedup for a single instance of the network. An implementation of our framework on faster FPGAs with emulation of multiple copies of the network can further increase the difference in runtime between software- and hardware-based approaches.

5. CONCLUSION

In this work, we proposed a methodology to implement logical models of signaling networks using FPGAs, in order to efficiently study these networks. We have shown that the speedup that an FPGA simulation provides over the classic simulation-based approach can be as high as 3,100X for a single model. Since multiple copies of the model can be implemented on a single FPGA (up to six in our current implementation) and run in parallel, the speedup scales approximately linearly with the number of copies (leading to 17,000X speedup in the Case study III). These results demonstrate the tremendous potential of hardware-based emulation.

As future work, we plan to create a general-purpose framework that can also translate models given in either logical rule-, reaction network- or reaction rule-based format into a faithful hardware implementation replica of the system that reproduces both dynamic and steady-state behavior. This will facilitate fast and accurate analysis of system behavior for many different initial conditions and scenarios, thus opening the possibility for scalable simulation and analysis of highly complex biological systems.

6. REFERENCES

- [1] Zhang, R., Shah, M. V., Yang, J., Nyland, S. B., Liu, X., Yun, J. K., Albert, R. and Loughran, T. P., Jr. Network model of survival signaling in large granular lymphocyte leukemia. *Proc Natl Acad Sci U S A*, 105, 42 (2008), 16308-16313.
- [2] Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81, 25 (1977), 2340-2361.
- [3] Novak, M. and Tyson, J. J. A model for restriction point control of the mammalian cell cycle. *Journal of Theoretical Biology*, 230, 4 (2004), 563-579.
- [4] Faeder, J. R., Blinov, M. L. and Hlavacek, W. S. Rule-based modeling of biochemical systems with BioNetGen. *Methods Mol Biol*, 500(2009), 113-167.
- [5] Albert, I., Thakar, J., Li, S., Zhang, R. and Albert, R. Boolean network simulations for life scientists. *Source Code Biol Med*, 3(2008), 16.
- [6] Saez-Rodriguez, J., Alexopoulos, L. G., Epperlein, J., Samaga, R., Lauffenburger, D. A., Klamt, S. and Sorger, P. K. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Mol Syst Biol*, 5(2009), -.
- [7] Bornholdt, S. Boolean network models of cellular regulation: prospects and limitations. *Journal of the Royal Society Interface*, 5(2008), S85-S94.
- [8] Thomas, R., Thieffry, D. and Kaufman, M. Dynamical behaviour of biological regulatory networks--I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bull Math Biol*, 57, 2 (1995), 247-276.
- [9] Kauffman, S. A. Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol*, 22, 3 (1969), 437-467.
- [10] Albert, R. and Othmer, H. G. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *J Theor Biol*, 223, 1 (2003), 1-18.
- [11] Davidich, M. I. and Bornholdt, S. Boolean Network Model Predicts Cell Cycle Sequence of Fission Yeast. *PLoS One*, 3, 2 (2008), -.
- [12] Sanchez, L. and Thieffry, D. A logical analysis of the *Drosophila* gap-gene system. *Journal of Theoretical Biology*, 211, 2 (2001), 115-1.
- [13] Salwinski, L. and Eisenberg, D. In silico simulation of biological network dynamics. *Nature Biotechnology*, 22, 8 (2004), 1017-1019.
- [14] Keane, J. F., Bradley, C. and Ebeling, C. A compiled accelerator for biological cell signaling simulations. In *Proceedings of the Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays* (2004), 233-241.
- [15] Yoshihara, M., Iwaoaka, Y., Nishikawa, Y., Kojima, T., Osana, Y., Shibata, Y., Iwanaga, N., Yamada, H., Kitano, H., Funahashi, A., Hiroi, N. and Amano, H. *FPGA Implementation of a Data-Driven Stochastic Biochemical Simulator with the Next Reaction Method*. City, 2007.
- [16] Garg, A., Di Cara, A., Xenarios, I., Mendoza, L. and De Micheli, G. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24, 17 (2008), 1917-1925.
- [17] ModelSim - Advanced Simulation and Debugging, Mentor Graphics, <http://model.com/>.
- [18] Brace, K. S., Rudell, R. L. and Bryant, R. E. Efficient implementation of a BDD package. In *Proceedings of the Proceedings of the 27th ACM/IEEE Design Automation Conference* (1990), 40-45.
- [19] Bjesse, P. and Claessen, K. SAT-Based Verification without State Space Traversal. In *Proceedings of the Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design* (2000), 372-389.
- [20] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. and Hwang, L. J. Symbolic model checking: states and beyond. *Inf. Comput.*, 98, 2 (1992), 142-170.
- [21] Mendoza, L. A network model for the control of the differentiation process in Th cells. *Biosystems*, 84, 2 (2006), 101-114.
- [22] Miskov-Zivanov, N., Turner, M., Morel, P. and Faeder, J., Title, 2011.
- [23] XILINX, <http://www.xilinx.com/>.